Agreement-Based Interactions for Experimental Science

Katarzyna Keahey,[1] Takuya Araki, [1,2] Peter Lane[1]

[1] Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439 USA
[2] NEC Internet Systems Research Laboratories, Kanagawa 216-8555, Japan
keahey@mcs.anl.gov, araki@mcs.anl.gov, lane@mcs.anl.gov

Abstract. Enabling quality of service in Grids requires not only resource management strategies but also the development of protocols enabling structured negotiation for the use of resources. In this paper, we describe design, implementation, and application of an agreement-based infrastructure. We then discuss its use in the virtual control room developed for the National Fusion Collaboratory.

1 Introduction

Computational Grids are of proven usefulness to scientific community. However, in order to make Grid computing a viable model for experimental science, capabilities enabling it to respond to QoS requirements, such as real-time execution, are needed. For example, the National Fusion Collaboratory (NFC) project [1] wants to enable scientists to run analysis codes on the Grid in the roughly 20-minute period between the pulses of a fusion experiment. In addition to developing sophisticated resource management strategies and algorithms, enabling this capability requires the development of protocols enabling structured negotiation for the use of resources.

An agreement-based services infrastructure combines information, negotiation, and execution services that allow clients to query the availability of a particular service in the context of their priority needs, as well as to compare offers from different providers. Once a service agreement is achieved, the service provider can use it to drive resource management. Combining subsidiary agreements allows for the creation of agreements of arbitrary complexity, and automatic management allows providers to both leverage and counteract the changing conditions on the Grid. We see this flexible formalism as underlying the future development of the Grid.

In this paper we describe an implementation of an agreement-based infrastructure based on the WS-Agreement specification [2] currently developed at the Global Grid Forum (GGF). We describe terms for specific applications including combined agreements, dependency-based agreements, and agreement templates. To manage uncertainty, we associate agreements with confidence levels representing the quality of agreement to the client. Finally, we demonstrate the use of our implementation in practice, working under to constraints of a virtual control room developed by the National Fusion Collaboratory for use in fusion experiments.

2 Agreements: Architecture and Implementation

In this section, we describe the architecture and implementation of our agreement negotiation structure used by the services described in Section 3.

2.1 Overview of WS-Agreement

WS-Agreement [2] is a draft specification developed at the GRAAP working group of the GGF describing a negotiation approach to service management. The negotiation process can be viewed as a discovery phase in which clients advertise their needs to the providers and the providers represent what capabilities they can provide. This phase ends when both sides commit. A complete agreement represents a concretization of use policy representing an agreed to relationship between a client and a provider.

According to the specification, agreements are represented as Grid services [3] and inherit all the properties of a Grid service. They are created by factories, subject to soft-state lifetime management, and enable access to state exposed as service data elements (SDEs). In particular, one of the SDEs exposes the agreement terms. A new agreement service can be requested by a client and created whenever a factory determines that the requested terms are suitable for beginning a negotiation. If the requested terms are unacceptable, an exception is returned.

The WS-Agreement specification defines a term type (`wsa:TermType`) for describing agreement terms, but it does not provide a term language for any domain. It is assumed that such term languages to describe domain-specific concepts (data transfer, resource management, application-specific concepts) will be developed separately through complementary work in other working groups.

The current focus of the working group is on architecture and the negotiation model. The negotiation model allows renegotiating agreements after creation and concludes in a commit stage that can be triggered by either side. The negotiation is fine grained and proceeds on the level of specific terms.

2.2 Our Implementation

We implemented agreement-based interactions using the Globus Toolkit 3 (GT3). While our implementation was heavily influenced by WS-Agreement [2] and Web Service Level Agreement (WSLA) [4], our use case did not require a full implementation of it. Instead, we focused on defining terms and functionality required by the application and practical experiences with the system.

Instead of representing each agreement as a service, the factory creates and maintains a table of current "agreement entries" exposed as factory SDEs and managed as factory state. Our negotiation process is simplified and emphasizes discovery. An agreement factory allows a client to retrieve an "agreement template" (based on the `AgreementTermType` in the section below) advertising some initial values of the agreements it supports (for example, a factory may support only services of a fixed description). The clients can then fill out some or all fields in this template and propose an agreement. By filling out more or fewer fields, the client can effectively ask a more or less concrete question about the availability of a specific service. The agreement may be rejected (if the terms specified by the client cannot be satisfied) by returning an exception. Alternatively, the factory can supply values for fields not filled out by the client and return it as provider's precommitted offer together with an agreement handle identifying the agreement. Precommitment on the provider's side results in creating an "agreement entry" with a short expiration time that can be extended if the client commits. After receiving factory response, the client can either commit to the proposed agreement or try again. Our negotiation model is simpler than WS-Agreement as it does not implement multi-phase negotiations or support renegotiation once an agreement has been created. Further, it allows negotiation on the level of the whole agreement only. We also support a simpler commitment model: only the provider can precommit and client commit. Client's commitment extends the agreement time to the end of availability time.

A committed agreement causes the factory to automatically instantiate the requisite application service when the availability period of the service described by the agreement starts. This is done to reduce the impact of service creation overhead on agreement claiming. The client can then obtain the handle to the application service from the factory and claim the agreement from the application service which triggers the execution of desired actions.

2.3 Agreement Term Type

An agreement represents a commitment that services described by the service description will be provided during a specified time of service availability with a specified QoS (whenever applicable). At most one such service will be provided at a time, but it may be claimed multiple times as the availability period allows. Our agreement terms are described as follows:

```
<xsd:complexType name="AgreementTermType">
    <xsd:sequence>
        <xsd:element name="parties" type="tns:AgreementPartiesType"/>
        <xsd:element name="serviceInstanceHandle" type="xsd:anyURI"/>
        <xsd:element name="dependency" type="xsd:anyURI"
                minOccurs="0"
                maxOccurs="unbound"/>
        <xsd:element name="availability" type="tns:ScheduleType"/>
        <xsd:element name="expirationTime" type="xsd:dateTime"/>
        <xsd:element name="serviceLevel" type="tns:serviceLevelType"/>
        <xsd:element name="serviceDescription" type="xsd:anyType"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AgreementPartiesType">
    <xsd:sequence>
        <xsd:element name="client" type="xsd:anyURI"/>
        <xsd:element name="provider" type="xsd:anyURI"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ScheduleType">
    <xsd:sequence>
```

```
        <xsd:element name="startTime" type="xsd:dateTime"/>
        <xsd:element name="endTime" type="xsd:dateTime"/><
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="serviceLevelType">
    <xsd:sequence>
        <xsd:element name="timeBound" type="xsd:duration"/>
        <xsd:element name="confidenceLevel" type="xsd:int"/>
    </xsd:sequence>
</xsd:complexType>
```

The first three items of the schema correspond to the `wsa:ContextType` of the specification. They describe the parties of the agreement and include the Grid Service Handle (GSH) of the client and the provider. The `serviceInstanceHandle` element holds the GSH of the application service created as a result of the agreement. The `dependency` element contains the agreement handle(s) which the agreement is dependent on (see Section 3.2 and Section 3.4 for illustration).

The `availability` element defines the time period when the services specified in the agreement are available. The `expirationTime` element corresponds to the lifetime of the agreement (not the created service).

The `serviceLevel` element refers to QoS guaranteed by the agreement. The `timeBound` element describes guaranteed execution time. While some entities can be managed in a deterministic fashion (CPU reservation for example), others are not (for example, data transfer over the Internet). In order to account for this uncertainty, the service provider accompanies the agreement terms with a confidence level with which it can provide the terms.

The `serviceDescription` element is a domain-specific element; its content depends on the service. Examples will be described in the next section.

3 Services

3.1 CPU Reservation Service

The CPU reservation service is used for reserving a CPU resource. The `serviceDescription` element is as follows:

```
<xsd:sequence>
    <xsd:element name="CPUUtilization" type="xsd:int"/>
    <xsd:element name="hostname" type="xsd:string"/>
</xsd:sequence>
```

Our assumption is that we will be reserving only one CPU per host and possibly timesharing that CPU with other jobs. Thus, CPUUtilization describes the percentage of CPU to be used. The implementation of the resource reservation service is similar to GARA [5]: to implement resource reservation, this service uses DSRT [6], which has functionality to allocate specified percentage of CPU cycles to a certain process. The service maintains a reservation table; when an agreement is proposed, the table is first consulted to make sure there is a slot available, and if a specified percentage of CPU cycles is free during the period of availability, the proposed agreement is accepted.

When the agreement is claimed, the job ID of a job to which the reservation should be applied is passed as an argument to the claim. Note, that the services description here is not related to QoS: the agreement simply states that a certain service will be provided.

3.2 Job Execution Service

The job execution service can be customized to represent the execution of a program. The service description represents the name of the program and concrete arguments. The `serviceDescription` element for this specific application is as follows:

```
<xsd:sequence>
    <xsd:element name="application" type="xsd:string"/>
    <xsd:element name="timeSteps" type="xsd:int"/>
    <xsd:element name="executionMode" type="xsd:string"/>
</xsd:sequence>
```

The agreement for this service guarantees execution of a specific service description with QoS. It is important that the service description externalize all the arguments that QoS may depend on; in this example both `executionMode` and `timeSteps` influence the execution time of our application.

To meet the QoS, this service reserves CPU resources using the CPU reservation service. In the current implementation, resource reservation is made by a job execution agreement factory when the level of the execution service is negotiated, but we also envision scenarios where the client can use a preexisting reservation as input to negotiation. The GSH of CPU reservation service is stored as the `dependency` element of `AgreementTermType`. Based on this reservation, the service uses prediction to estimate execution time for the service and its confidence level modeled as prediction error. It should be noted that the agreement does not indicate in what way or to what extent the service depends on the dependency agreement; the knowledge of how to "consume" the dependency in terms of both estimating confidence levels and claiming the dependency is application specific and contained in the implementation of agreement service and application service respectively. Another point to note is that although the terms of the agreement are based on resource management, they are to some extent advisory; that is, the estimate of execution time is based on prediction rather than adaptive management of the application.

When the agreement is claimed, the service starts executing the job using GT3's Grid Resource and Allocation Manager (GRAM) service. The CPU is claimed by associating the job ID of the job started in this way with the reservation.

3.3 Data Transfer Service

The data transfer service is implemented based on the GT3 reliable file transfer (RFT) [7] service and uses RFT's `transferRequest` as part of its service description. Among other qualities, `transferRequest` contains information about the source and destination of the transfer, needed to calculate QoS. The exact parameters are as follows:

```
<xsd:sequence>
    <xsd:element name="transferRequest"
        type="rft-types:TransferRequestType"/>
    <xsd:element name="size" type="xsd:int"/>
</xsd:sequence>
```

Estimates of execution time (transfer time, in this case) are based on prediction depending on historical data for this transfer and confidence level on associated error.

Since fusion codes produce multiple files as a result of a run, the data transfer service has been customized to operate on directories of data rather than individual files: the data is tarred before RFT is invoked and untarred at destination.

3.4 Workflow Service

The workflow service is used to provide end-to-end application service by coordinating several subsidiary services. In our case the workflow is very simple and consists of application execution and data transfer of output data. The `serviceDescription` of Workflow Service is as follows:

```
<xsd:sequence>
    <xsd:element name="application" type="xsd:string"/>
    <xsd:element name="timeSteps" type="xsd:int"/>
    <xsd:element name="executionMode" type="xsd:string"/>
    <xsd:element name="outputDestination" type="xsd:string"/>
</xsd:sequence>
```

As before, the service description externalizes arguments on which the QoS depends; in this case we add the argument describing the destination of the data to those on job execution.

The workflow service directly depends on the job execution and data transfer services for guarantees given to its QoS. We currently store those dependencies as the dependency element of `AgreementTermType,` although in principle a stronger dependency should be expressed: the workflow service depends on having these services executed in sequence.

Negotiating a composite agreement based on a workflow is more complex as it requires the workflow service in turn to negotiate subsidiary agreements. Further, dependencies between the elements of the workflow may impose an order on negotiating subsidiary agreements. The end-to-end time is calculated by combining execution times of the services in ways appropriate to workflow (in our case by adding, but in general we could use min/max) and using the confidence level of subsidiary services to calculate a weighted error. As with the other services, the workflow and its subsidiary services are instantiated when the availability period starts. Claiming an agreement on an application service will trigger claims on subsidiary services.

4 Case Study: Interactions in the Virtual Control Room

Our prototype infrastructure and services were put to the test in the virtual control room experiment at SC03 illustrating how Grids can be used in fusion science experiments. Fusion experiments operate in a pulsed mode, producing plasmas of up to 10 seconds duration every 15 to 20 minutes, with multiple pulses per experiment. Decisions for changes to the next plasma pulse are made by analyzing measurements from the previous plasma pulse (hundreds of megabytes of data) within roughly 15 minutes between pulses. This mode of operation could be made more efficient by the ability to leverage Grid resources to do more analysis and simulation in the short time between pulses. Hence, the ability to do time-bounded execution in the Grids is of critical importance.

The virtual control room experiment followed the script of typical experiment preparation and interaction. Before an experiment, a scientist can negotiate an agreement for the execution of a remote fusion code and request for data to be delivered to a specific location. This process allows the scientist to experiment with, and fine-tune the parameters for the execution of the code. Thus, the agreement-based system is used not only to perform management actions but also to structure and automate experimental process that has grown more complex with the use of Grids.

The agreement formed in this way promises to deliver an end-to-end QoS on execution time of the service as long as the execution is requested within a certain availability window. Delivering the QoS entails combining data transfers with application execution and CPU management. At the time of the experiment, the client can request service execution against a previously formed agreement and expect it to be satisfied with the agreed on QoS.

In the experiment our implementation and services discussed earlier were used to obtain agreements and claim execution of the to the EFIT code performing equilibrium fitting on the results of fusion experiment. The code runs were initiated from the SC03 show floor in Phoenix, Arizona; EFIT execution was performed at Princeton Plasma Physics Lab (PPPL); and the end results were transferred to the control room team at General Atomics in California.

	Job Execution Service	Data Transfer Service	Total Execution Time
Measured	95 sec	54 sec	173 sec
Agreement	95 sec, 90%	53 sec, 93%	172 sec, 92%

The table above shows how our actual execution values compared with what was promised in the agreement. The “measured” row shows the mean of 10 values for each quality measured. The “agreement” row shows promised value and the level of confidence with which it is promised. The results show good agreement with estimated values. The overhead is large mainly because, while the time spent on the respective services was measured locally, the end-to-end execution time was measured from the SC show floor, accumulating the high latencies of acknowledgement messages from the services. Nevertheless, the overall execution time was satisfactory and the infrastructure deemed acceptable for experimental use.

5 Conclusions and Future Work

Although our implementation provides only a simple negotiation model, we found that it fulfilled the needs of our use case very well. The negotiation phase worked well as a discovery capability customized to the needs of a client. In fact, some of our current agreements are used in "advisory" capacity and enable the scientist to do, in a structured way, what was previously done in an ad hoc manner: estimate times for codes that will be run during the experiment. Underpinning this interaction are the resource management actions ensuring the success of such preparations.

Given the dynamic and unreliable nature of a Grid environment, any guarantee must be qualified: resources may become unavailable, or policies and priorities may change at any moment. Furthermore, while some qualities in the Grid can be managed (CPU reservations, for example), others cannot: we cannot reserve bandwidth on the Internet or predict exactly the runtime of an application. For this reason, we have introduced *levels of confidence* used by the provider to represent the strength of a QoS guarantee. We modeled them as the probability that a certain QoS will be achieved. While this measure is correct from a provider's perspective, it is not very helpful for the client because it does not give the client the means of verifying failure rate. With the addition of resource management, however, it is possible to convert a provider's failure rate into a failure rate for a specific user. Such a guarantee would be more appropriate from the perspective of our use case.

Finally, the potential of agreements stems from the fact that they constitute a target to guide adaptive actions. More research will have to be done to fulfill their promise.

Acknowledgments

This work was supported by the Mathematical, Information and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, SciDAC Program, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38.

References

[1. Keahey, K., T. Fredian, Q. Peng, D.P. Schissel, M. Thompson, I. Foster, M. Greenwald, and D. McCune, *Computational Grids in Action: the National Fusion Collaboratory.* Future Generation Computing Systems (to appear), October 2002. **18**(8): p. 1005-1015.

2. Czajkowski, K., A. Dan, J. Rofrano, S. Tuecke, and M. Xu, *Agreement-based Grid Service Management (OGSI-Agreement) Version 0.*

https://forge.gridforum.org/projects/graap-wg/document/Draft_OGSI-Agreement_Specification/en/1/Draft_OGSI-Agreement_Specification.doc, 2003.

3. Foster, I., C. Kesselman, J. Nick, and S. Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. 2002: Open Grid Service Infrastructure WG, Global Grid Forum,.
4. Ludwig, H., A. Keller, A. Dan, and R.P. King, *A Service Level Agreement Language for Dynamic Electronic Services.* IBM Research Report RC22316 (W0201-112), January 24, 2002.
5. Foster, I., C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. *A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation*. in *Proc. International Workshop on Quality of Service*. 1999.
6. Nahrstedt, K., H. Chu, and S. Narayan. *QoS-aware Resource Management for Distributed Multimedia Applications*. in *Journal on High-Speed Networking, IOS Press*. December 1998.
7. Madduri, R., C. Hood, and W. Allcock, *Reliable File Transfer in Grid Environments.* LCN, 2002: p. 737-738.